Logical methods in NLP 2012

Preliminaries

Michael Moortgat

Abstract

Natural languages exhibit dependency patterns that are provably beyond the recognizing capacity of context free grammars. In recent research, a family of grammar formalisms has emerged that gracefully deals with such phenomema beyond context-free and at the same time keeps a pleasant (polynomial) parsing complexity.

We study some key formalisms in this so-called 'mildly context-sensitive' family, together with the cognitive interpretation of the kind of dependencies they express. We look at the dependency structures projected by grammatical derivations.

Background reading. Chapter 2 from Laura Kallmeyer, Parsing Beyond Context-Free Grammars. Springer, Cognitive Technologies, 2010. Chapters 3 to 6 from Marco Kuhlmann, Dependency Structures and Lexicalized Grammars. Springer.

More to explore. A standard reference for the general theory is Lewis & Papadimitriou, Elements of the theory of computation.

1. Formal grammars

A grammar is a tuple (V, Σ, R, S) with

- V is an alphabet;
- Σ a subset of V, a finite set of terminal symbols;
- R a set of rules, a finite subset of $V^* \times V^*$

  we write $\alpha \longrightarrow \beta$

  with $\alpha, \beta \in V^*$ (strings over terminals/non-terminals)
- S an element of $V - \Sigma$, the start symbol

Putting restrictions on the form of the production rules leads to a hierarchy of formal grammars, each with their own expressivity and complexity properties.

Chomsky hierarchy

$$R \subset CF \subset CS \subset RE$$

type	language	automaton	restrictions
3	regular	finite state automaton	$A \longrightarrow w$; $A \longrightarrow wB$
2	context-free	push-down automaton	$A \longrightarrow \gamma$
1	context-sensitive	linear bounded automaton	$\alpha A \beta \longrightarrow \alpha\gamma\beta$, $\gamma \neq \epsilon$
0	recursively enumerable	Turing machine	$\alpha \longrightarrow \beta$

(notation: A, B for nonterminals, w for a string of terminals, α, β as before)

Adding fine-structure

R and CF have shown to be extremely useful for capturing NL patterns.

- R: speech,phonology, morphology
- CF: the larger part of NL syntax

CS is too expressive to be informative about the limitations of the language faculty.

$\rightsquigarrow$ let's impose a finer granularity to chart the territory between CF and CS.

Regular languages, finite state automata

We have characterized grammars for regular languages as a restricted form of CFG. There is a more natural, direct characterization.

Regular expressions Concatenation, choice, repetition

$$E ::= a \mid 1 \mid 0 \mid EE \mid E + E \mid E^*$$

Deterministic finite state automaton a 5-tuple $M = (K, \Sigma, \delta, q_0, F)$ with

K a finite set of states,

$q_0 \in K$ the initial state,

$F \subseteq K$ the set of final states,

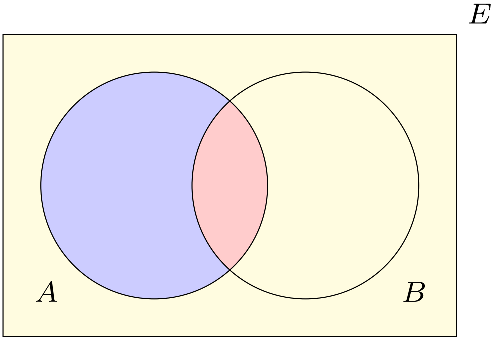
Σ an alphabet of input symbols,

δ, the **transition function**, is a function from $K \times \Sigma$ to K.

Non-deterministic: transition relation

Regular patterns: semantic automata

Consider examples of the form 'all poets dream', 'not all politicians can be trusted', in general: QAB



To understand the Q words it suffices to compare

- blue: $A - B$
- red: $A \cap B$

Tree of numbers

A triangle with pairs (n, m), for growing numbers of A:

- $n : |A - B|$
- $m : |A \cap B|$

$$
\begin{array}{lc}
|A| = 0 & (0,0) \\
|A| = 1 & (1,0)\ (0,1) \\
|A| = 2 & (2,0)\ (1,1)\ (0,2) \\
|A| = 3 & (3,0)\ (2,1)\ (1,2)\ (0,3) \\
|A| = 4 & (4,0)\ (3,1)\ (2,2)\ (1,3)\ (0,4) \\
|A| = 5 & (5,0)\ (4,1)\ (3,2)\ (2,3)\ (1,4)\ (0,5) \\
\ldots & \ldots
\end{array}
$$

Tree of numbers

A triangle with pairs (n, m), for growing numbers of A:

- $n : |A - B|$
- $m : |A \cap B|$

$$
\begin{array}{lc}
|A| = 0 & (0,0) \\
|A| = 1 & (1,0)\ (0,1) \\
|A| = 2 & (2,0)\ (1,1)\ (0,2) \\
|A| = 3 & (3,0)\ (2,1)\ (1,2)\ (0,3) \\
|A| = 4 & (4,0)\ (3,1)\ (2,2)\ (1,3)\ (0,4) \\
|A| = 5 & (5,0)\ (4,1)\ (3,2)\ (2,3)\ (1,4)\ (0,5) \\
\dots & \dots
\end{array}
$$

Example: all A B

Patterns: all, no, some, not all

+
− +
− − +
− − − +
− − − − +
− − − − − +

all

+
+ −
+ − −
+ − − −
+ − − − −
+ − − − − −

no

−
− +
− + +
− + + +
− + + + +
− + + + + +

some

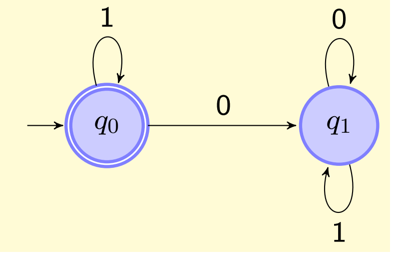
−
+ −
+ + −
+ + + −
+ + + + −
+ + + + + −

not all

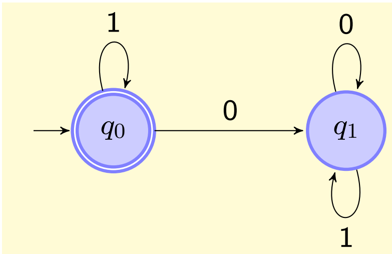
Q words as semantic automata

A Q automaton runs on a string of 0's and 1's: 0 for elements in $A - B$, 1 for elements in $A \cap B$. Acceptance of a string means that QAB holds.

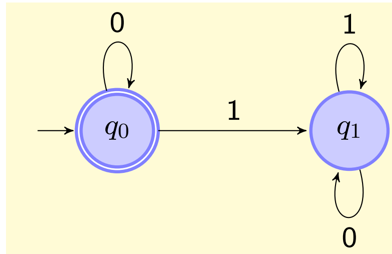
Example: all A B



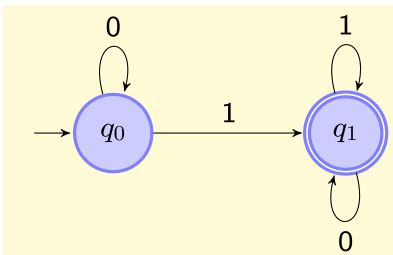
Automata: all, no, some, not all



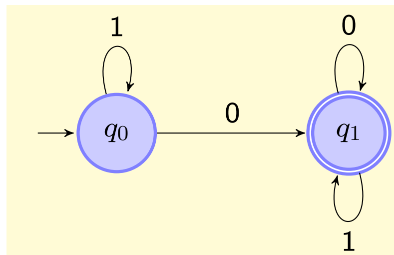
all



no



some



not all

Beyond R

How do we know a language is not regular?

Pumpability We say a string w in language L is *k-pumpable* if there are strings $u_0, \ldots, u_k$ and $v_1, \ldots, v_k$ satisfying

$$w = u_0 v_1 u_1 v_2 u_2 \ldots u_{k-1} v_k u_k$$

$$v_1 v_2 \ldots v_k \neq \epsilon$$

$$u_0 v_1^i u_1 v_2^i u_2 \ldots u_{k-1} v_k^i u_k \in L \quad \text{for every } i \geq 0$$

Theorem Let L be an infinite regular language. Then there are strings x, y, z such that $y \neq \epsilon$ and $xy^i z \in L$ for each $i \geq 0$ (i.e. 1-pumpability)

Example The language $L = \{a^n b^n \mid n \geq 0\}$ is not regular. (Compare $a^* b^*$)

Context-free grammars

A **context-free grammar** G is a 4-tuple (V, Σ, R, S), where

V is an alphabet,

Σ (the set of **terminals**) is a subset of V,

R (the set of **rules**) is a finite subset of $(V - \Sigma) \times V^*$, and

S (the **start symbol**) is an element of $V - \Sigma$.

The members of $V - \Sigma$ are called **nonterminals**.

Push-down automata

A **push-down automaton** is a 6-tuple $M = (K, \Sigma, \Gamma, \Delta, q_0, F)$ with

K a finite set of states,

$q_0 \in K$ the initial state,

$F \subseteq K$ the set of final states,

Σ an alphabet of input symbols,

Γ an alphabet of stack symbols,

$\Delta \subseteq (K \times \Sigma^* \times \Gamma^*) \times (K \times \Gamma^*)$ the transition relation.

Acceptance, non-determinism

We say that

$$((q, u, \beta), (q', \gamma)) \in \Delta$$

if the machine, in state q with β on top of the stack, can read u from the input tape, replace β by γ on top of the stack, and enter state q'.

When different such transitions are simultaneously applicable, we have a non-deterministic **pda**.

A **pda** accepts a string $w \in \Sigma^*$ iff from the configuration (q_0, w, ϵ) there is a sequence of transitions to a configuration $(q_f, \epsilon, \epsilon)$ ($q_f \in F$) — a final state with end of input and empty stack.

PDA example: deterministic

Automaton M for $L = \{wcw^R \mid w \in \{a,b\}^*\}$. Let $M = (K, \Sigma, \Gamma, \Delta, q_0, F)$, with $K = \{q_0, q_1\}, \Sigma = \{a, b, c\}, \Gamma = \{a, b\}, F = \{q_1\}$, and Δ consists of the following transitions:

1. $((q_0, a, \epsilon), (q_0, a))$
2. $((q_0, b, \epsilon), (q_0, b))$
3. $((q_0, c, \epsilon), (q_1, \epsilon))$
4. $((q_1, a, a), (q_1, \epsilon))$
5. $((q_1, b, b), (q_1, \epsilon))$

Sample run

Run of M on the string $lionoil$:

K	INPUT	STACK	Δ
q_0	$lionoil$	ϵ	PUSH
q_0	$ionoil$	l	PUSH
q_0	$onoil$	il	PUSH
q_0	$noil$	oil	
q_1	oil	oil	POP
q_1	il	il	POP
q_1	l	l	POP
q_1	ϵ	ϵ	

Corresponding CFG

Context-free grammar G with $L(G) = \{wcw^R \mid w \in \{a,b\}^*\}$. Let $G = (V, \Sigma, R, S)$ with

$$
\begin{array}{rcl}
V & = & \{S, a, b, c\} \\
\Sigma & = & \{a, b, c\} \\
R & = & \{ \; S \longrightarrow aSa, \\
 & & \quad\; S \longrightarrow bSb, \\
 & & \quad\; S \longrightarrow c \qquad \}
\end{array}
$$

PDA: non-deterministic

Automaton M for $L = \{ww^R \mid w \in \{a,b\}^*\}$. Let $M = (K, \Sigma, \Gamma, \Delta, q_0, F)$, with $K = \{q_0, q_1\}, \Sigma = \Gamma = \{a, b\}, F = \{q_1\}$, and Δ consists of the following transitions:

1. $((q_0, a, \epsilon), (q_0, a))$
2. $((q_0, b, \epsilon), (q_0, b))$
3. $((q_0, \epsilon, \epsilon), (q_1, \epsilon))$
4. $((q_1, a, a), (q_1, \epsilon))$
5. $((q_1, b, b), (q_1, \epsilon))$

Compare transition (3) with the earlier deterministic example. In state q_0, the machine can make a choice: push the next input symbol on the stack, or jump to q_1 without consuming any input.

Semantic automata: beyond regular

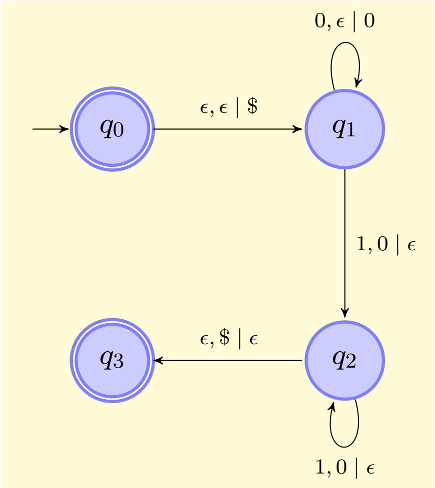
Van Benthem's THEOREM: the 1st order definable Q words are precisely the quantifying expressions recognized by permutation-invariant acyclic finite automata.

But . . . there are Q words that require stronger computational resources.

Example: most A B here we need a stack memory.

INPUT	STACK
0 0 1 0 1 1 1	
0 1 0 1 1 1	0
1 0 1 1 1	0 0
0 1 1 1	0
1 1 1	0 0
1 1	0
1	
. . .	. . .

Abstract example: $0^n 1^n$



Compare after reading a 1, a finite automaton would have forgotten how many 0's it has seen.

Beyond CFG

CF pumping theorem Let G be a context-free grammar generating an infinite language. Then there is a constant k, depending on G, so that for every string w in $L(G)$ with $|w| \geq k$ it holds that $w = xv_1yv_2z$ with

- $|v_1v_2| \geq 1$
- $|v_1yv_2| \leq k$
- $w = xv_1^iyv_2^iz \in L(G)$, for every $i \geq 0$

This is 2-pumpability.

Example $L = \{a^nb^nc^n \mid n \geq 0\}$ is not context-free.

Example Patterns of the w^2 type in Dutch/Swiss German (Huijbregts, Shieber):

... dat Jan Marie de kinderen zag leren zwemmen

Mild context-sensitivity

Challenge An emergent thesis underlining the cognitive relevance of the above: 'Human cognitive capacities are constrained by polynomial time computability' (Frixione, Minds and Machines; Szymanyk, etc). The challenge then becomes: Can we step beyond CF without losing the attractive computational properties?

Joshi's program A set of languages $\mathcal{L}$ is **mildly context-sensitive** iff

- $\mathcal{L}$ contains all CFL
- $\mathcal{L}$ recognizes a bounded amount of cross-serial dependencies:

  there is $n \geq 2$ such that $\{w^k \mid w \in \Sigma^*\} \in \mathcal{L}$ for all $k \leq n$
- The languages in $\mathcal{L}$ are polynomially parsable
- The languages in $\mathcal{L}$ have the constant growth property

Constant growth holds for **semilinear** languages.

Semilinearity

Parikh mapping Let $X = \{a_1, \ldots, a_n\}$ be an alphabet with some fixed order on the elements. The Parikh mapping $p : X^* \longrightarrow \mathbb{N}^n$ is defined as follows:

- for all $w \in X^*$, $p(w) \doteq \langle |w|_{a_1}, \ldots, |w|_{a_n} \rangle$

  where $|w|_{a_i}$ is the number of occurrences of a_i in w
- for all $L \subseteq X^*$, $p(L) \doteq \{p(w) \mid w \in L\}$ is the Parikh image of L

Letter equivalence Two words are l.e. if they contain an equal number of occurrences of each terminal symbol; two languages are l.e. if every string in one is l.e. to a string in the other and v.v.

Semilinearity A language is semilinear iff l.e. to a regular language.

Parikh's theorem All context-free languages are semilinear.

Closure properties

The following are useful tools to abstract away from irrelevant details of the 'linguistic phenomena'.

String homomorphism For two alphabets Σ_1, Σ_2, a function $f : \Sigma_1^* \longrightarrow \Sigma_2^*$ iff for all $v, w \in \Sigma_1^*$: $f(vw) = f(v)f(w)$.

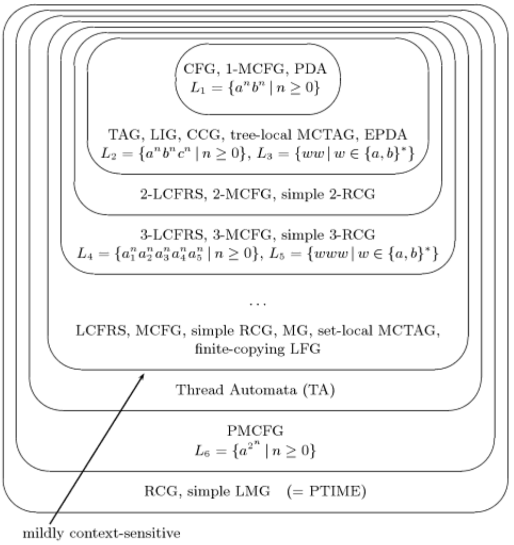
Note that h is determined by its values on single alphabet symbols. Note also that h is allowed to erase material: for nonempty w, $h(w)$ may be empty.

Closure under homomorphisms given Σ_1, Σ_2, for every context-free language L_1 over Σ_1 and every homomorphism $f : \Sigma_1^* \longrightarrow \Sigma_2^*$, $h(L_1) = \{h(w) \mid w \in L_1\}$ is a context-free language.

Closure under intersection with regular languages for every context-free language L and every regular language R, $L \cap R$ is a context-free language.

The landscape beyond context-free

Below, from Kallmeyer's book, the hierarchy of mildly context-sensitive formalisms and some characteristic patterns.



2. Dependency structures

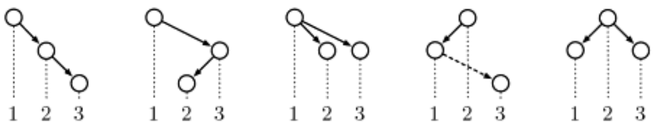
Marco Kuhlmann, Dependency Structures and Lexicalized Grammars.

Aim to systematically relate expressivity/complexity of grammar formalisms to structural properties of the dependency graphs induced by the derivations of these formalisms.

Dependency structures trees with a total order on their nodes. Two relations:

- governance: $u \trianglelefteq v$, u governs v, v depends on u
- precedence: $u \preceq v$

Visualization



(a) D_1 (b) D_2 (c) D_3 (d) D_4 (e) D_5

Dependency structures and grammars

Classes

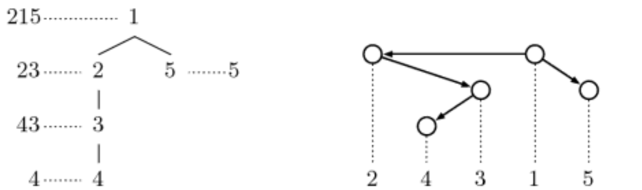
- $\mathcal{D}_1$ projective dependency structures: all yields form an interval
- $\mathcal{D}_k$ dependency structures of bounded degree: measures number of detached parts
- $\mathcal{D}_{wn}$ well-nested dependency structures: non-crossing partitions

Below the classes of dependency structures induced by the derivations of a number of grammar formalisms.

formalism	class
Context-free Grammar	$\mathcal{D}_1$
Linear Context-free Rewriting Systems $\text{LCFRS}(k)$, also $\text{MCFG}(k)$	$\mathcal{D}_k$
Coupled Context-free Grammars $\text{CCFG}(k)$	$\mathcal{D}_k \cap \mathcal{D}_{wn}$
Tree Adjoining Grammars TAG	$\mathcal{D}_2 \cap \mathcal{D}_{wn}$

$\mathcal{D}_1$ projective dependency structures

K establishes a bijection between $\mathcal{D}_1$ and the set of all treelet-ordered trees (each node annotated with a total order on that node and its children).



(a) tree (b) D_8 (treelet-order traversal)

```
TREELET-ORDER-COLLECT(u)
L ← NIL
foreach v in order[u]
      do if v = u
            then L ← L · [u]
            else  L ← L · TREELET-ORDER-COLLECT(v)
return L
```

$\mathcal{D}_1$ and context-free derivations

A grammar is lexicalized if each rule introduces exactly one terminal (called the anchor of that rule). Example (for $a^n b^n$)

$$S \longrightarrow a\ S\ B \mid a\ B \quad ; \quad B \longrightarrow b$$

Induced dependency structures Let G be a lexicalized CFG and $t \in \mathrm{Term}_{\Sigma(G)}$ a derivation tree. The dependency structure induced by t is the structure $D = (\mathrm{nodes}(t), \trianglelefteq, \preceq)$ where

- $u \trianglelefteq v$ iff u dominates v in t
- $u \preceq v$ iff u precedes v in $[\![t]\!]$ (the evaluation of t in the linearization semantics for G)

Correspondence $\mathcal{D}(CFG) = \mathcal{D}_1$. Derivations of lexicalized CFGs induce projective dependency structures.

$\mathcal{D}_1$ enumerative combinatorics

The number of projective dependency structures over n nodes is counted by integer sequence https://oeis.org/A006013:

$$1, 2, 7, 30, 143, 728, 3876, 21318, 120175, 690690, 4032015, 23841480, \ldots$$

with generating formula

$$\binom{3n+1}{n} / (n+1)$$

where $\binom{n}{k}$ (the binomial coefficient) has initial values $\binom{n}{0} = 1$ for all $n \in \mathbb{N}$ and $\binom{0}{k} = 0$ for integers $k > 0$; the recursive case for $n, k > 0$ is

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Working session We try to gain a clearer understanding of the combinatorics by recasting $\mathcal{D}_1$ in terms of binary trees.

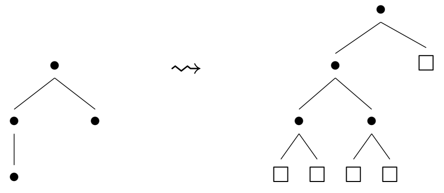
- step 1: encoding general trees as bintrees
- step 2: read off projective linearization from bintrees

From general to binary trees

First child-next sibling binary trees We write n'_i for the node of the binary tree b corresponding to node n_i of the general tree t. The root of t is mapped to the root of b; then

- if n_l is the leftmost child of n_k in t, n'_l is the left child of n'_k in b
- if n_s is the next sibling of n_k in t, n'_s is the right child of n'_k in b

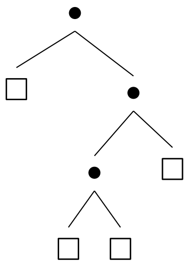
Example writing $\square$ for empty daughters in the binary representation



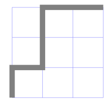
Binary trees: semantics

n node binary trees have nice interpretations, including

- Dyck words: well-nested strings of n pairs of parentheses
- Monotonic paths on $n \times n$ grid



$()(())$



Binary trees: enumerative combinatorics

The sequence of Catalan numbers C_n counts the number of n-node binary trees:

$$C_n \quad = \quad \frac{1}{n+1}\binom{2n}{n} \quad = \quad \binom{2n}{n} - \binom{2n}{n+1}$$

This is integer sequence http://oeis.org/A000108:

$$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$$

The recurrence below calculates C_{n+1} in terms of C_n:

$$C_0 = 1 \quad ; \quad C_{n+1} \quad = \quad \sum_{i=0}^{n} C_i C_{n-i}$$

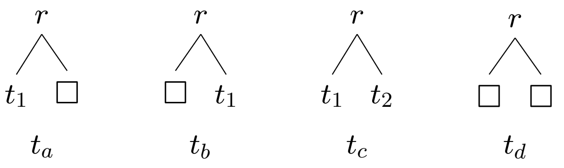
Challenge Find a recurrence relation for the number of n-node projective dependency structures based on C_n ...

Binary trees: projective dependency semantics

Relational pseudocode reading off projective dependency structures from an n-node binary tree:

lin Tree ListIn $\mathit{ListOut}$

with initialization $\mathit{ListIn} : 0 \ldots (n-1)$, $\mathit{ListOut} : []$



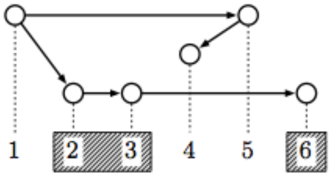
- lin t_a $\vec{u}$ $(r : \vec{v}) \leftarrow$ convex $\vec{u}$, select r $\vec{u}$ $\vec{u}'$, lin t_1 $\vec{u}'$ $\vec{v}$
- lin t_b $(r : \vec{u})$ $(r : \vec{v}) \leftarrow$ lin t_1 $\vec{u}$ $\vec{v}$
- lin t_c $\vec{u}'\vec{u}''$ $(r : \vec{v}\,\vec{v}') \leftarrow$ convex $\vec{u}'$, select r $\vec{u}'$ $\vec{u}'''$, lin t_1 $\vec{u}'''$ $\vec{v}$, lin t_2 $\vec{u}''$ $\vec{v}'$
- lin t_d r r

Beyond $\mathcal{D}_1$

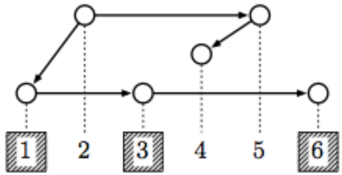
Projectivity and beyond:

- projectivity: every subtree spans an interval
- gap-degree k: every subtree has at most k gaps (=block degree $k+1$)
- well-nestedness: disjoint edges must not overlap

Block/gap degree



(a) D_1, block-degree 2



(b) D_2, block-degree 3

The block-degree of $S \subseteq A$ wrt a chain $(A; \preceq)$ is the cardinality of $S/\equiv_S$.

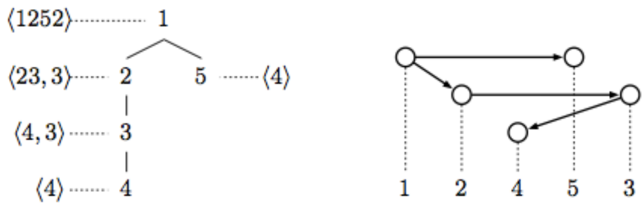
$\equiv_S$ coarsest congruence relation on S: $a \equiv_S b$ iff for all $c \in [a, b]$, $c \in S$.

Gap degree: block degree minus 1.

Traversal of block-ordered trees

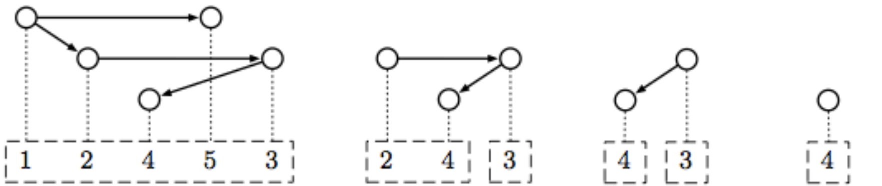
```
BLOCK-ORDER-COLLECT(u)
L ← NIL; calls[u] ← calls[u] + 1
foreach v in order[u][calls[u]]
        do if v = u
              then L ← L · [u]
              else  L ← L · BLOCK-ORDER-COLLECT(v)
return L
```

Illustration (Correct annotation for node 5 to $\langle 5 \rangle$ ...)



(a) tree (b) D_3 (block-order traversal)

Segmented dependency structures



(a) $D_4 \cong D_3$ (b) $D_4/2$ (c) $D_4/3$ (d) $D_4/4$

Definition 4.2.1 Let $D = (V\,; \trianglelefteq, \preceq)$ be a dependency structure, and let $\equiv$ be a congruence relation on D. The *segmentation* of D by $\equiv$ is the structure $D' := (V\,; \trianglelefteq, \preceq, R)$, where R is a new ternary relation on V defined as follows:

$$(u, v_1, v_2) \in R \;:\Longleftrightarrow\; v_1 \equiv v_2 \;\wedge\; \forall w \in [v_1, v_2].\; w \in \lfloor u \rfloor\,.$$

The elements of the set $V/{\equiv}$ are called the *segments* of D'. □

Linearization

For u a node in a segmented dependency structure D, the set of blocks of u is the set $\lfloor u \rfloor / \equiv_u$.

> **Definition 4.2.3** Let T be a tree, and let $k \in \mathbb{N}$. A *linearization* of T with k components is a k-tuple $L = \langle \vec{u}_i \mid i \in [k] \rangle$ such that $\vec{u} := \vec{u}_1 \cdots \vec{u}_k$ is a list of the nodes of T in which each node occurs exactly once. The segmented dependency structure *induced* by a linearization L of T is the structure in which the governance relation is isomorphic to T, the precedence relation is isomorphic to $\vec{u}$, and the segments are isomorphic to the tuple components of L. □

Correspondence (compare: treelet-ordered trees and projective D)

- for every segmented D there is exactly one block-ordered tree T such that $D = dep(T)$.
- if T is a block-ordered tree in which each node is annotated with at most k lists, then $dep(T)$ is a segmented dependency structure with block degree at most k

Dependency structure algebras

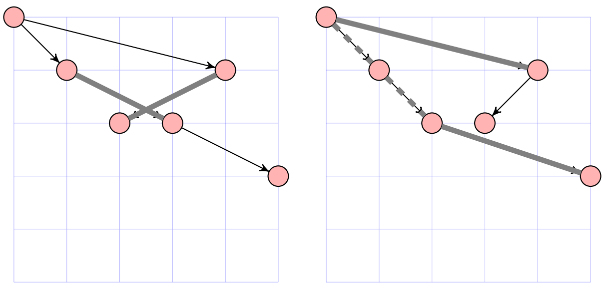
tbd

Well-nestedness

D is well-nested if for all edges $v_1 \to v_2$, $w_1 \to w_2$ in D it holds that

$$\text{if } v_1 \to v_2,\ w_1 \to w_2 \text{ overlap, then } v_1 \trianglelefteq w_1 \text{ or } w_1 \trianglelefteq v_1$$

Illustration



D_1 D_2

- D_1 ill-nested: edges $1 \to 3$ and $4 \to 2$ disjoint, overlapping;
- D_2 well-nested: edges $0 \to 4$ and $2 \to 5$ overlap, but $0 \trianglelefteq 2$

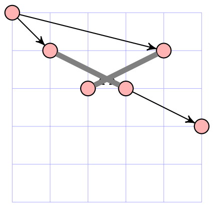
Well-nestedness and non-crossing partitions

A dependency structure D is well-nested iff for every node u of D, the set of constituents of u is non-crossing wrt the chain $(\lfloor u \rfloor; \preceq_{\lfloor u \rfloor})$

A partition Π on a chain $(A; \preceq)$ is non-crossing if whenever there exist $a_1 \prec b_1 \prec a_2 \prec b_2$ in A such that a_1, a_2 belong to the same class of Π and b_1, b_2 belong to the same class of Π, then these two classes coincide.

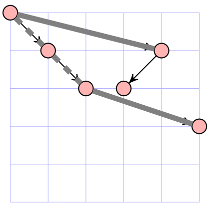
The set of constituents of a node u in D is $\{\{u\} \cup \{\lfloor v \rfloor \mid u \to v\}\}$.

Illustration Compare the constituents of node 0 in D_1 and D_2



D_1

$\{\{0\}, \{1, 3, 5\}, \{2, 4\}\}$

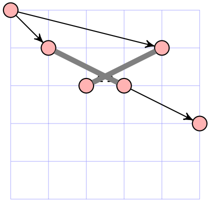


D_2

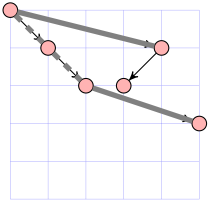
$\{\{0\}, \{1, 2, 5\}, \{3, 4\}\}$

Non-crossing partitions

Partitions induced by the constituents of node 0 in D_1 and D_2



D_1



D_2

$\{\{0\}, \{1, 3, 5\}, \{2, 4\}\}$ $\{\{0\}, \{1, 2, 5\}, \{3, 4\}\}$